

On computing the 2-vertex-connected components of directed graphs

Raed Jaberⁱ*

*Faculty of Computer Science and Automation, Technische Universität Ilmenau,
98694 Ilmenau, Germany*

Abstract

In this paper we consider the problem of computing the 2-vertex-connected components (2-vccs) of directed graphs. We present two new algorithms for solving this problem. The first algorithm runs in $O(mn^2)$ time, the second in $O(nm)$ time. Furthermore, we show that the old algorithm of Erusalimskii and Svetlov runs in $O(nm^2)$ time. In this paper, we investigate the relationship between 2-vccs and dominator trees. We also present an algorithm for computing the 3-vertex-connected components (3-vccs) of a directed graph in $O(n^3m)$ time, and we show that the k -vertex-connected components (k -vccs) of a directed graph can be computed in $O(mn^{2k-3})$ time. Finally, we consider three applications of our new algorithms, which are approximation algorithms for problems that are generalization of the problem of approximating the smallest 2-vertex-connected spanning subgraph of 2-vertex-connected directed graph.

Keywords: Graph algorithms, 2-vertex-connected components, Strong articulation points, Approximation algorithms

1. Introduction

Let $G = (V, E)$ be a directed graph with $|V| = n$ vertices and $|E| = m$ edges. A strong articulation point of G is a vertex whose removal increases the number of strongly connected components of G . A directed graph $G = (V, E)$ is said to be

*Tel.: +49 3677 69 -2786; fax: +49 3677 69 -1237.

Email address: raed.jaberi@tu-ilmenau.de (Raed Jaberⁱ)

k -vertex-connected if it has at least $k+1$ vertices and the induced subgraph on $V \setminus X$ is strongly connected for every $X \subset V$ with $|X| < k$. Thus, a strongly connected graph $G = (V, E)$ is 2-vertex-connected if and only if it has at least 3 vertices and it contains no strong articulation points. The 2-vertex-connected components of a strongly connected graph G are its maximal 2-vertex-connected subgraphs. The concept was defined in [6]. For more information see [12].

In 2010, Georgiadis [8] gave a linear time algorithm to test whether a strongly connected graph G is 2-vertex-connected or not. Later, Italiano et al. [12] gave a linear time algorithm for the same problem which is faster in practice than the algorithm of Georgiadis [8]. Moreover, the algorithm of Italiano et al. [12] can find all the strong articulation points of a directed graph G in linear time. The algorithm from [12] solved an open problem posed by Beldiceanu et al. (2005) [3]. In 1980, Erusalimskii and Svetlov [6] gave an algorithm for computing the 2-vccs of a directed graph, whose running time was not analyzed. In this work we show that this algorithm runs in $O(nm^2)$ time. Furthermore, we present two new algorithms for computing the 2-vccs of a directed graph. The first algorithm runs in $O(mn^2)$ time, and the second in $O(nm)$ time. The question posed by Italiano et al. [12] whether the problem is solvable in linear time still remains open.

The remainder of this paper is organized as follows. In section 2, we briefly describe the algorithm of Italiano et al. [12] for finding the strong articulation points of a directed graph. In section 3, we briefly describe the algorithm of Erusalimskii and Svetlov [6] for computing the 2-vccs of a directed graph and analyze its running time. In section 4, we present a new algorithm for computing the 2-vccs that contain a certain vertex if such a component exists. Then we use this algorithm to compute all the 2-vccs of a directed graph in $O(mn^2)$ time. In section 5, we present another new algorithm for computing all the 2-vccs of a directed graph in $O(nm)$ time. Afterwards, we investigate the relationship between 2-vccs and dominator trees in section 6. In section 7, we present an algorithm for computing the 3-vccs of a directed graph in $O(n^3m)$ time, and we show that the k -vccs of a directed graph can be computed in $O(mn^{2k-3})$ time. Finally in section 8, we consider three applications of our new algorithms, which are approximation algorithms for problems that are generalization of the problem of approximating the smallest 2-vertex-connected spanning subgraph of 2-vertex-connected directed graph.

2. The algorithm of Italiano et al.

In this section, we briefly describe the algorithm of Italiano et al. [12] for computing all the strong articulation points of a directed graph. This algorithm will be used later. The content of this section is based on [12]. We first explain some definitions and notations which will be used throughout this paper. A *flowgraph* $G(v) = (V, E, v)$ is a directed graph with $|V| = n$ vertices, $|E| = m$ edges, and a distinguished start vertex $v \in V$ such that every vertex $w \in V$ is reachable from v . For a flowgraph $G(v) = (V, E, v)$, the *dominance relation* of $G(v)$ is defined as follows: a vertex $w \in V$ is a *dominator* of vertex $u \in V$ if every path from v to u includes w . By $dom(w)$ we denote the set of dominators of vertex w . Obviously, the set of dominators of the start vertex in $G(v)$ is $dom(v) = \{v\}$. For every vertex $w \in V$ with $w \neq v$, $\{v, w\}$ is a subset of $dom(w)$; we call w, v the *trivial dominators* of w . A vertex u is a *non-trivial dominator* in $G(v)$ if there is some $w \notin \{v, u\}$ such that $u \in dom(w)$. The set of all non-trivial dominators is called $D(v)$. The dominance relation is reflexive, transitive, and antisymmetric. A vertex $u \in V$ is an *immediate dominator* of vertex $w \in V$ in $G(v)$ if $u \in dom(w)$ and all elements of $dom(w) \setminus \{w\}$ are dominators of u . Every vertex w of $G(v)$ except the start vertex v has a unique immediate dominator, which is denoted by $imd(w)$. The edges (u, w) , where u is the immediate dominator of w , form a tree with root v , called the *dominator tree* of $G(v)$, denoted by $DT(v)$. Vertex $w \in V$ is a dominator of vertex $u \in V$ in $G(v)$ if and only if w is an ancestor of u in $DT(v)$.

Let $G = (V, E)$ be a directed graph. Let F be a subset of E and let U be a subset of V . We use $G \setminus F$ to denote the directed graph obtained from G by deleting all the edges in F . We use $G \setminus U$ to denote the directed graph obtained from G by removing all the vertices in U and their incident edges. By $G[F]$ we denote the directed graph $(V[F], F)$ whose $V[F] = \{w \mid \exists u \in V : (w, u) \in F \text{ or } (u, w) \in F\}$. By $G[U]$ we denote the directed graph $(U, E[U])$ whose $E[U] = \{(w, u) \mid w, u \in U \text{ and } (w, u) \in E\}$. $G[F]$ and $G[U]$ are subgraphs of G . The reversal graph of G is the directed graph $G^R = (V, E^R)$, where $E^R = \{(w, u) \mid (u, w) \in E\}$.

Let $G = (V, E)$ be a strongly connected graph and let v be a vertex in G . Since G^R is strongly connected, $G^R(v) = (V, E^R, v)$ is a flowgraph. By $D^R(v)$ we denote the set of all non-trivial dominators in the flowgraph $G^R(v)$. The algorithm of Italiano et al. is based on the following fact.

Fact 2.1. [12] *Let $G = (V, E)$ be a strongly connected graph, and let v be any vertex in G . Then vertex $w \in V$ with $w \neq v$ is a strong articulation point if and only if w is a non-trivial dominator in the flowgraph $G(v)$ or in the flowgraph $G^R(v)$.*

The set of the strong articulation points of an arbitrary directed graph G is the union of the strong articulation points of its strongly connected components. Algorithm 2.2 shows the algorithm of Italiano et al. [12]. More information on this algorithm can be found in [12, 7].

Algorithm 2.2 (SAVs(G)). (from [12])

Input: A strongly connected graph $G = (V, E)$.

Output: The strong articulation points of G .

- 1 Choose $v \in C$ arbitrarily.
- 2 **if** $G \setminus \{v\}$ is not strongly connected **then output** v .
- 3 Compute and **output** $D(v)$.
- 4 Calculate the reversal graph G^R .
- 5 Compute and **output** $D^R(v)$.

Fact 2.3. [12] *Algorithm 2.2 runs in $O(n + m)$ time.*

3. Algorithm of Erusalimskii and Svetlov

In this section, we briefly describe the algorithm of Erusalimskii and Svetlov [6] for computing the 2-vccs of a directed graph, and we analyze its running time. The latter analysis was missing in [6]. The content of this section is based mainly on [6]. In [6], the authors provided an algorithm for computing all *biblocks* of a directed graph, where the biblocks of a directed graph are its maximal strongly connected subgraphs that do not contain any strong articulation point. A biblock is either a 2-vcc, a single vertex or two vertices which are connected by two antiparallel edges. In this paper we are only interested in computing the 2-vccs of a directed graph. Let $G = (V, E)$ be a strongly connected graph, and let v be a strong articulation point in G . Then the vertex v does not necessarily occur in two or more 2-vccs of G [6]. Moreover, 2-vccs have the following property:

Fact 3.1. [6] *Let C_1^{2vc}, C_2^{2vc} be distinct 2-vccs in directed graph $G = (V, E)$. Then C_1^{2vc} and C_2^{2vc} have at most one vertex in common.*

In [6], the authors studied a class of directed graphs L defined as follows: A directed graph $G = (V, E)$ belongs to class L if it satisfies the following conditions:

1. If C_1, C_2, \dots, C_t are the strongly connected components of G , then there are no edges between C_i and C_j for distinct $i, j \in \{1, \dots, t\}$.

2. For every strong articulation point v the directed graph $G \setminus \{v\}$ satisfies (1).

Let $G = (V, E)$ be a directed graph. By $U(G)$ we denote the undirected graph formed from G by deleting the directions of the edges. In [6], the following was proved:

Fact 3.2. [6] *Let $G = (V, E) \in L$ be a directed graph. The vertices of the 2-vccs of G coincide with the vertices of the 2-connected components (i.e. 2-vertex-connected components [5]) of the undirected version $U(G)$.*

The main idea behind the algorithm of Erusalimskii and Svetlov [6] is as follows. Given a directed graph $G = (V, E)$, the algorithm computes a directed graph $G' \in L$ such that the 2-vccs of G coincide with the 2-vccs of G' . Then all 2-vccs of G' can be easily computed by using Fact 3.2. Algorithm 3.3 shows this algorithm [6]:

Algorithm 3.3 (ErusalimskiiSvetlov(G)). (from [6])

Input: A directed graph $G = (V, E)$.

Output: The 2-vccs of G .

```

1  Repeat
2      Compute the strongly connected components of  $G$ .
3      Remove from  $G$  the edges between the strongly connected components of  $G$ .
4      for every vertex  $v \in V$  do
5          Compute the strongly connected components of  $G \setminus \{v\}$ .
6          Remove from  $G$  the edges between the strongly connected
7          components of  $G \setminus \{v\}$ .
8  until no edge was removed during step 6.
9  We obtain a directed graph  $G' \in L$ .
10 Compute the 2-connected components  $C_1^{2vc}, C_2^{2vc}, \dots, C_k^{2vc}$  of  $U(G')$ .
12 Output  $C_1^{2vc}, C_2^{2vc}, \dots, C_k^{2vc}$ .
```

Fact 3.4. [6] *Let $G = (V, E)$ be a directed graph and let $G' \in L$ be the directed graph obtained after running algorithm 3.3 on the input graph G (in step 9), then the 2-vccs of G coincide with the 2-vccs of G' .*

Theorem 3.5. *The running time of algorithm 3.3 is $O(nm^2)$.*

Proof. The number of iterations of the repeat-loop is at most m since at least one edge is removed in each iteration. The strongly connected components of a directed graph can be found in linear time using Tarjan's algorithm [13]. In each iteration of

the repeat-loop, steps 4–7 require $O(n(n + m))$ time. The 2-connected components of an undirected graph can be computed in linear time using Tarjan’s algorithm [13]. Thus, the total running time of algorithm 3.3 is $O(m(n(m + n))) = O(nm^2)$. \square

4. Computing 2-vertex-connected components that contain a certain vertex

In this section, we present a new algorithm for computing all the 2-vccs of a directed graph $G = (V, E)$ that contain a certain vertex $v \in V$. Note that it can happen that a vertex is not contained in any 2-vcc, as Figure 1 illustrates. We may

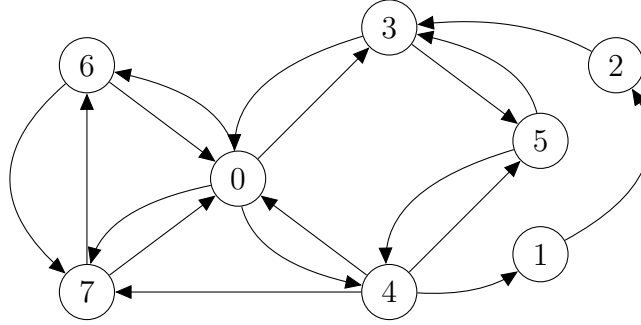


Figure 1: Vertices 1, 2 do not lie in any 2-vcc.

always assume that G is strongly connected.

Let $G = (V, E)$ be a strongly connected graph and let v be an arbitrary vertex in G . We consider the dominator tree $DT(v)$ of the flowgraph $G(v) = (V, E, v)$. By $K(v)$ we denote the set of direct successors of the root v in the dominator tree $DT(v)$. A vertex $w \in V$ belongs to the set $K(v)$ if and only if $(v, w) \in E$ or there exist two vertex-disjoint paths from v to w in G . Let $G^R = (V, E^R)$ be the reversal graph of G . We consider the dominator tree $DT^R(v)$ of $G^R(v)$ and denote by $K^R(v)$ the set of direct successors of the root v in $DT^R(v)$.

Lemma 4.1. *Let $G = (V, E)$ be a strongly connected graph and let v be an arbitrary vertex in G . Then only elements of $((K(v) \cap K^R(v)) \cup \{v\})$ can belong to the 2-vccs which contain the vertex v .*

Proof. Let $w \in V$, and assume that $w \notin (K(v) \cap K^R(v)) \cup \{v\}$, i.e. $w \notin K(v) \cap K^R(v)$ and $w \neq v$. Then $w \in (V \setminus (K(v) \cup \{v\})) \cup (V \setminus (K^R(v) \cup \{v\}))$. There are two cases to consider:

1. $w \notin K(v) \cup \{v\}$. Then there exists a non-trivial dominator s such that every path from v to w includes s in G . Therefore, there are no two vertex-disjoint paths from v to w in G .
2. $w \notin K^R(v) \cup \{v\}$. We argue as in case (1).

□

Lemma 4.2. *Let $G = (V, E)$ be a directed graph and let v be a strong articulation point in G . Let C^{2vc} be a 2-vcc of G with $v \in C^{2vc}$. Then all vertices of $C^{2vc} \setminus \{v\}$ lie in a strong connected component C of $G \setminus \{v\}$, i.e. $C^{2vc} \setminus \{v\} \subseteq C$.*

Proof. Since C^{2vc} is a 2-vcc of G , the directed graph $G[C^{2vc}]$ does not contain any articulation point. Thus, $G[C^{2vc} \setminus \{v\}]$ is strongly connected. Moreover, $G[C^{2vc} \setminus \{v\}]$ is a subgraph of $G \setminus \{v\}$. Consequently, $C^{2vc} \setminus \{v\}$ is a subset of a strongly connected component of $G \setminus \{v\}$. □

Now we can describe our algorithm for computing the 2-vccs of a directed graph $G = (V, E)$ that contain v .

Algorithm 4.3 (2VCCsAlgorithm1(G)).

Input: A directed graph $G = (V, E)$ and a vertex $v \in V$.

Output: The 2-vccs that contain v .

- 1 $G = (V, E) \leftarrow$ the strongly connected component C_v of G with $v \in C_v$.
- 2 **if** G is 2-vertex-connected **then**
- 3 **Output** V .
- 4 **else if** v is not a strong articulation point in G and $|K(v) \cap K^R(v)| \geq 2$ **then**
- 5 Recursively compute the 2-vccs of $G[(K(v) \cap K^R(v)) \cup \{v\}]$ that contain v and **output** them.
- 6 **else if** v is a strong articulation point in G **then**
- 7 Compute the strongly connected components of $G \setminus \{v\}$.
- 8 **for** every strongly connected component C of $G \setminus \{v\}$ **do**
- 9 **if** $G[C \cup \{v\}]$ is strongly connected and $|C| \geq 2$ **then**
- 10 Recursively compute the 2-vccs of $G[C \cup \{v\}]$ that contain v .
- 11 **Output** all the computed 2-vccs.

Algorithm 4.3 works as follows. First, line 1 finds the strongly connected component C_v of G with $v \in C_v$ using Tarjan's algorithm [13] and assigns the directed graph $G[C_v]$ to G because all the 2-vertex-connected components which contain v

lie in $G[C_v]$. Then, line 2 tests whether G is 2-vertex-connected using the algorithm of Italiano et al. [12], and if it is, line 3 outputs V . Otherwise, the algorithm tests whether v is a strong articulation point in G or not. If v is not a strong articulation point in G and $|K(v) \cap K^R(v)| \geq 2$, then line 5 recursively computes the 2-vccs of $G[(K(v) \cap K^R(v)) \cup \{v\}]$ that contain v and outputs them. This is correct by Lemma 4.1. If v is a strong articulation point in G , then the for loop of lines 8–10 recursively computes the 2-vccs of $G[C \cup \{v\}]$ that include v for every strongly connected component C of $G \setminus \{v\}$, where $G[C \cup \{v\}]$ is strongly connected and $|C| \geq 2$. This is correct by Lemma 4.2.

Theorem 4.4. *Algorithm 4.3 runs in $O(nm)$ time.*

Proof. The dominators of a flowgraph can be found in linear time [2, 1]. The strong articulation points of a directed graph can also be computed in linear time using the algorithm of Italiano et al. [12]. Furthermore, the strongly connected components of a directed graph can be computed in linear time using Tarjan’s algorithm [13]. At each level of the recursion at least one vertex must be removed in lines 4–5 or the set of vertices must be split in lines 6–10. Hence, the recursion depth is at most n . Fix some recursion level. We consider the cost of the calls of the procedure excepting the recursion. For one call, the cost is linear in the size of the current subgraph. Let $G'[C_1 \cup \{v\}] = (V_1, E_1), G'[C_2 \cup \{v\}] = (V_2, E_2), \dots, G'[C_t \cup \{v\}] = (V_t, E_t)$ be the subgraphs of the directed graph $G' = (V', E')$ considered on this level in all calls. Then $\sum_{1 \leq i \leq t} |E_i| \leq |E'|$ since the strongly connected components of G' are disjoint. The total cost at each level of the recursion is therefore $O(m)$. \square

Corollary 4.5. *Let $G = (V, E)$ be a directed graph. If we apply algorithm 4.3 for every vertex $v \in V$, we can find all the 2-vccs of G in $O(n^2m)$ time.*

Proof. This follows immediately from Theorem 4.4. \square

5. Computing 2-vertex-connected components of directed graphs

In this section, we present a new algorithm for computing all the 2-vccs of a directed graph in $O(nm)$ time. Our algorithm is based on the following Lemma.

Lemma 5.1. *Let $G = (V, E)$ be a directed graph and let w be a strong articulation point in G . Let C^{2vc} be a 2-vcc of G . Then all vertices of $C^{2vc} \setminus \{w\}$ lie in a strongly connected component C of $G \setminus \{w\}$, i.e. $C^{2vc} \setminus \{w\} \subseteq C$.*

Proof. As in Lemma 4.2. □

Corollary 5.2. *Let $G = (V, E)$ be a directed graph and let w be a strong articulation point in G . The 2-vccs of G lie in the subgraphs $G[C_1 \cup \{w\}]$, $G[C_2 \cup \{w\}]$, \dots , $G[C_t \cup \{w\}]$, where C_1, C_2, \dots, C_t are the strongly connected components of $G \setminus \{w\}$.*

We now describe our algorithm for computing all the 2-vccs of a directed graph $G = (V, E)$.

Algorithm 5.3 (2VCCsAlgorithm2(G)).

Input: A directed graph $G = (V, E)$.

Output: The 2-vccs of G .

```

1  if  $G$  is 2-vertex-connected then
2      Output  $V$ .
3  else
4      Find a strong articulation point  $w$  of  $G$ .
5      Compute the strongly connected components of  $G \setminus \{w\}$ .
6      for each strongly connected component  $C$  of  $G \setminus \{w\}$  do
7          Recursively compute the 2-vccs of  $G[C \cup \{w\}]$  and output them.
```

Theorem 5.4. *Algorithm 5.3 runs in $O(nm)$ time.*

Proof. The strong articulation points of a directed graph can be computed in linear time using the algorithm of Italiano et al. [12]. The strongly connected components of a directed graph can also be computed in linear time using Tarjan's algorithm [13]. Let C_1, C_2, \dots, C_t be the strongly connected components of $G \setminus \{w\}$. Clearly, the edge sets of the subgraphs $G[C_1 \cup \{w\}]$, $G[C_2 \cup \{w\}]$, \dots , $G[C_t \cup \{w\}]$ are disjoint. Thus the total cost at each recursion level is $O(m)$. Since the vertex set of a graph in a recursive call is smaller than the original vertex set, the recursion depth is at most n . Thus the total time is $O(nm)$. □

6. The relationship between 2-vertex-connected components and dominator trees

In this section, we prove a connection between the 2-vccs of a directed graph and dominator trees.

Theorem 6.1. *Let $G = (V, E)$ be a strongly connected graph and let v be an arbitrary vertex in G . Let C^{2vc} be a 2-vcc of G . Then either all elements of C^{2vc} are direct successors of some vertex $w \notin C^{2vc}$ or all elements $C^{2vc} \setminus \{w\}$ are direct successors of some vertex $w \in C^{2vc}$ in the dominator tree $DT(v)$ of the flowgraph $G(v)$.*

Proof. We consider two cases:

1. $v \in C^{2vc}$. In this case, all elements of $C^{2vc} \setminus \{v\}$ are direct successors of v in $DT(v)$ since for every vertex x of $C^{2vc} \setminus \{v\}$, there are two vertex-disjoint paths from v to x in $G[C^{2vc}]$, hence in $G(v)$.
2. $v \notin C^{2vc}$. Then there is a vertex $x \in C^{2vc}$ such that $imd(x) = w$ and $w \notin C^{2vc}$. We show, by contradiction, that w is a dominator for every vertex of C^{2vc} . Assume that there is some vertex $y \in C^{2vc}$, $y \neq x$ such that $w \notin dom(y)$. Consequently, there exists a path p from v to y not containing w . This path enters C^{2vc} in vertex $u \in C^{2vc}$. This means that the vertices of p from v to u are outside of C^{2vc} . Moreover, there are two vertex-disjoint paths from u to x in $G[C^{2vc}]$. Thus, there is a path from v to x not containing w . Therefore, we have $w \notin dom(x)$, which contradicts that $imd(x) = w$. Now we consider two cases:

- a) All paths from w to x are completely outside of $G[C^{2vc}]$. Then x is a dominator of all vertices $y \in C^{2vc}$. (Assume that there is a path from v to y avoiding x . By the above, w is on this path. We can extend p inside C^{2vc} to reach x , contradicting the assumption all paths from w to x are completely outside of $G[C^{2vc}]$.)

For every vertex $y \in C^{2vc}$, x is the immediate dominator of y in $DT(v)$, since there are two vertex-disjoint paths from x to y .

- b) There are at least two vertex-disjoint paths p_1, p_2 from w to x such that p_1 enters C^{2vc} in vertex y and p_2 enters C^{2vc} in vertex y' with $y \neq y'$. Since there are a path from y to y' and a path from y' to y in $G[C^{2vc}]$, there are two vertex-disjoint paths from w to y and two vertex-disjoint paths from w to y' in $G(v)$. Therefore, the vertices y, y' are direct successors of w in $DT(v)$. Now we prove that every vertex $z \in C^{2vc} \setminus \{x, y, y'\}$ is also direct successor of w . There are two case to consider:

- (i) All paths from y to z and all paths from y' to z have a vertex $z' \in C^{2vc}$ in common with $z' \notin \{y, y', z\}$. Consequently, all the paths from y to z contain z' in $G[C^{2vc}]$, where $z' \notin \{y, z\}$. Hence, z' is a strong articulation point in $G[C^{2vc}]$ by [12, Lemma 2.1] of Italiano et al., which contradicts that $G[C^{2vc}]$ is 2-vcc of G .
- (ii) To interrupt all paths from $\{y, y'\}$ to z , one has to remove at least two vertices. We add a vertex $s \notin V$ and two edges $(s, y), (s, y')$ to G . Clearly,

s and z are not adjacent. A separator of all paths from s to z is a set of vertices whose removal interrupts all paths from s to z . A minimal separator of all paths from s to z has two vertices. By Menger's Theorem (1927) there are two vertex-disjoint paths from s to z . Thus, there exist a path p from y to z and a path p' from y' to z in $G[C^{2vc}]$ such that p, p' are vertex-disjoint. As a consequence, there are two vertex-disjoint paths from w to z in $G(v)$. Therefore, z is direct successor of w in $DT(v)$.

□

By $M(w)$ we denote the set of direct successors of vertex w in the dominator tree of a flowgraph. Algorithm 6.2 shows a new algorithm for computing all the 2-vccs of a strongly connected graph G using Theorem 6.1.

Algorithm 6.2 ($\text{All2VsCCs}(G)$).

Input: A strongly connected graph $G = (V, E)$.

Output: The 2-vccs of G .

```

1  if  $G$  is 2-vertex-connected then
2      Output  $V$ .
3  else
4      Compute the strong articulation points of  $G$ .
5      Choose a vertex  $v \in V$  that is not a strong articulation point of  $G$ .
6      Compute the dominator trees  $DT(v)$  and  $DT^R(v)$ .
7      Choose a dominator tree of  $\{DT(v), DT^R(v)\}$  that contains more
8          non-trivial dominators.
9      for each vertex  $w \in V$  do
10         if  $|M(w)| \geq 2$  then
11             if  $G[M(w) \cup \{w\}]$  is not strongly connected then
12                 Compute the strongly connected components of  $G[M(w) \cup \{w\}]$ .
13                 for each strongly connected component  $C$  of  $G[M(w) \cup \{w\}]$  do
14                     if  $|C| \geq 3$  then
15                         Recursively compute the 2-vccs of  $G[C]$  and output them.
16                 else
17                     Recursively compute the 2-vccs of  $G[M(w) \cup \{w\}]$  and output them.
```

Algorithm 6.2 works as follows. First, line 1 tests whether the strongly connected graph G is 2-vertex-connected using the algorithm of Italiano et al. [12], and if it is, line 2 outputs V . Otherwise, the algorithm finds a dominator tree whose depth is at

least 2 as follows. Line 5 chooses a vertex $v \in V$ which is not a strong articulation point of G . Then, line 6 computes the dominator trees $DT(v)$ and $DT^R(v)$ since at least one of them has non-trivial dominators. In order to reduce the recursion depth, we choose a dominator tree of $\{DT(v), DT^R(v)\}$ that contains more non-trivial dominators. $M(w)$ is the set of direct successors of vertex w in the dominator tree that is chosen in line 7. For each vertex $w \in V$ with $|M(w)| \geq 2$, the algorithm tests whether if $G[M(w) \cup \{w\}]$ is strongly connected, and if it is, line 17 recursively computes the 2-vccs of $G[M(w) \cup \{w\}]$. Otherwise, the for loop of lines 13–15 recursively computes the 2-vccs of $G[C]$ for every strongly connected component C of $G[M(w) \cup \{w\}]$.

Theorem 6.3. *Algorithm 6.2 runs in $O(nm)$ time.*

Proof. Let v, w be distinct vertices in G . Then the edge sets of the subgraphs $G[M(v) \cup \{v\}]$, $G[M(w) \cup \{w\}]$ are disjoint since these subgraphs have at most one vertex in common. The rest of the proof is similar to the proof of Theorem 5.4. \square

7. Computing 3-vertex-connected components of a directed graph

The k -vertex-connected components of a directed graph are its maximal k -vertex-connected subgraphs. This definition is a natural generalization of 2-vccs which are defined by Italiano et al. [12]. In this section, we present an algorithm for computing the 3-vccs of a directed graph. Our algorithm is based on the following Lemma, which is the obvious generalization of Lemma 5.1.

Lemma 7.1. *Let $G = (V, E)$ be a 2-vertex-connected directed graph and let $X \subset V$ be a vertex-cut of G with $|X| = 2$. Let C^{3vc} be a 3-vcc of G . Then all vertices of $C^{3vc} \setminus X$ lie in a strongly connected component C of $G \setminus X$, i.e. $C^{3vc} \setminus X \subseteq C$.*

Proof. Because C^{3vc} is a 3-vcc of G , the directed graph $G[C^{3vc}]$ does not contain any vertex-cut $Y \subset V$ with $|Y| < 3$ by definition. Hence, $G[C^{3vc} \setminus X]$ is strongly connected. Furthermore, $G[C^{3vc} \setminus X]$ is a subgraph of $G \setminus X$. Therefore, $C^{3vc} \setminus X$ is a subset of a strongly connected component C of $G \setminus X$, i.e. $C^{3vc} \setminus X \subseteq C$. \square

Corollary 7.2. *Let $G = (V, E)$ be a 2-vertex-connected directed graph and let $X \subset V$ be a vertex-cut of G with $|X| = 2$. The 3-vccs of G lie in the subgraphs $G[C_1 \cup X]$, $G[C_2 \cup X]$, \dots , $G[C_t \cup X]$, where C_1, C_2, \dots, C_t are the strongly connected components of $G \setminus X$.*

Algorithm 7.3 ($3VCCs(G)$).

Input: A directed graph $G = (V, E)$.

Output: The 3-vccs of G .

```

1  if  $G$  is 3-vertex-connected then
2      Output  $V$ .
3  else if  $G$  is 2-vertex-connected then
4      Find a vertex-cut  $X$  of  $G$ .
5      Compute the strongly connected components of  $G \setminus X$ .
6      for each strongly connected component  $C$  of  $G \setminus X$  do
7          Recursively compute the 3-vccs of  $G[C \cup X]$  and output them.
8  else
9      Compute the 2-vccs of  $G$ .
10     for each 2-vcc  $C^{2vc}$  of  $G$  do
11         Recursively compute the 3-vccs of  $G[C^{2vc}]$  and output them.

```

Algorithm 7.3 shows our algorithm for computing the 3-vccs of a directed graph G . This algorithm works as follows. First, line 1 tests whether the directed graph G is 3-vertex-connected using Gabow's algorithm [10], and if it is, line 2 outputs V . Otherwise, the algorithm tests whether G is 2-vertex-connected using the algorithm of Italiano et al. [12]. If G is 2-vertex-connected, then line 4 finds a vertex-cut X of G using Gabow's algorithm [10] and the for loop of lines 6–7 recursively computes the 3-vccs of $G[C \cup X]$ for each strongly connected component C of $G \setminus X$. This is correct by Corollary 7.2. If G is neither 3-vertex-connected nor 2-vertex-connected, then line 9 computes the 2-vccs of G using Algorithm 5.3 and the for loop of lines 10–11 recursively computes the 3-vccs of $G[C^{2vc}]$ for each 2-vcc C^{2vc} of G .

Theorem 7.4. *Algorithm 7.3 runs in $O(n^3m)$ time.*

Proof. Let G be a directed graph. The vertex connectivity κ and a corresponding separator in G can be found using Gabow's algorithm [10] in $O((n + \min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$ time. Furthermore, 2-vertex-connectivity can be tested using the algorithm of Italiano et al. [12] in linear time. Let C_1, C_2, \dots, C_t be the strongly connected components of $G \setminus X$ (see lines 5–7). Since $|X| = 2$, we have $|E[X]| < 3$. The edge sets of the subgraphs $G[C_1 \cup X] \setminus E[X], G[C_2 \cup X] \setminus E[X], \dots, G[C_t \cup X] \setminus E[X]$ are disjoint. By Theorem 5.4, the 2-vccs $C_1^{2vc}, C_2^{2vc}, \dots, C_t^{2vc}$ of G can be computed in $O(nm)$ time. Moreover, by Fact 3.1, the edge sets of the subgraphs $G[C_1^{2vc}], G[C_2^{2vc}], \dots, G[C_t^{2vc}]$ are disjoint (see lines 10–11). Let G_1, G_2, \dots, G_l be the subgraphs which are considered at any level of the recursion, let n_i be the number of vertices of G_i and let m_i

be the number of edges of G_i , where $1 \leq i \leq l$. Then, the total cost at each recursion level is $n_1 m_1 + n_2 m_2 + \dots + n_l m_l \leq (n_1 + n_2 + \dots + n_l) m \leq n^2 m$ since $n_i \leq n$ and $l \leq n$. Since the vertex set of a graph in a recursive call is smaller than the original vertex set, the recursion depth is at most n . Thus the total time is $O(n^3 m)$. \square

It is not difficult to see that any two k -vccs of a directed graph have at most $k - 1$ vertices in common. We can compute the k -vccs of a directed graph using the following Lemma, which is the obvious generalization of Lemma 7.1.

Lemma 7.5. *Let $G = (V, E)$ be a $(k - 1)$ -vertex-connected directed graph and let $X \subset V$ be a vertex-cut of G with $|X| = k - 1$. Let C^{kvc} be a k -vcc of G . Then all vertices of $C^{kvc} \setminus X$ lie in a strongly connected component C of $G \setminus X$, i.e. $C^{kvc} \setminus X \subseteq C$.*

Proof: The proof is similar to the proof of Lemma 7.1. \square

Theorem 7.6. *The k -vccs of a directed graph can be computed in $O(mn^{2k-3})$ time.*

Proof. We can modify Algorithm 7.3 by replacing “2-vertex-connected” with “ $(k - 1)$ -vertex-connected” and by replacing “3-vertex-connected” with “ k -vertex-connected”. The modified algorithm can compute the k -vccs of a directed graph. Its running time is bounded by the product of the recursion depth, n times the cost for computing $(k - 1)$ -vccs and vertex-cut. One can easily prove by induction on k that the running time of the modified algorithm is $O(mn^{2k-3})$. \square

8. Applications

In this section, we consider three applications of the new algorithms.

Problem 8.1. *Given a directed graph $G = (V, E)$, find a minimum cardinality set $E^* \subseteq E$ such that the 2-vccs of G coincide with the 2-vccs of the graph $G^* = (V, E^*)$.*

Clearly, the smallest 2-vertex-connected spanning subgraph of a 2-vertex-connected directed graph is a special case of problem 8.1 when G is 2-vertex-connected. Therefore, by the results from [11, 9] problem 8.1 is NP-hard.

Lemma 8.2. *There is a 1.5 approximation algorithm for problem 8.1 with running time $O(nm)$.*

Proof. First, we compute all the 2-vccs $C_1^{2vc}, C_2^{2vc}, \dots, C_t^{2vc}$ of the directed graph G using Algorithm 5.3. The edges of the set $E \setminus (E[C_1^{2vc}] \cup E[C_2^{2vc}] \cup \dots \cup E[C_t^{2vc}])$ are irrelevant. Let E_{opt} an optimal solution for problem 8.1. Then, by Fact 3.1, we have $E_{opt} = E_1 \cup E_2 \cup \dots \cup E_t$, where E_i an optimal solution for the subgraph $G[C_i^{2vc}]$. Let $opt = |E_{opt}|$ and $opt_i = |E_i|$. Then, $opt = \sum_{1 \leq i \leq t} opt_i$. In 2000, Cheriyan and Thurimella [4] gave a $(1 + 1/k)$ -approximation algorithm for the problem of finding a minimum-size k -vertex-connected spanning subgraph of a directed graph with m edges. This algorithm runs in $O(km^2)$ time. In 2011, Georgiadis [9] improved the running time of the algorithm of Cheriyan und Thurimella from $O(m^2)$ to $O(m\sqrt{n} + n^2)$ for $k = 2$. This improved algorithm [9] preserves the 1.5 approximation guarantee of the Cheriyan-Thurimella algorithm for $k = 2$. Let E'_i be an edge set obtained by running the improved algorithm [9] on the subgraph $G[C_i^{2vc}]$. Then, we have $\sum_{1 \leq i \leq t} |E'_i| \leq 1.5 \sum_{1 \leq i \leq t} opt_i \leq 1.5opt$ because the edge sets of $G[C_i^{2vc}]$, $1 \leq i \leq t$, are disjoint. The total running time is $O(\sum_{1 \leq i \leq t} (|E[C_i^{2vc}]| \sqrt{|C_i^{2vc}|} + |C_i^{2vc}|^2) + nm) = O(nm)$ because $\sum_{1 \leq i \leq t} (|E[C_i^{2vc}]| \sqrt{|C_i^{2vc}|} + |C_i^{2vc}|^2) \leq \sqrt{n} \sum_{1 \leq i \leq t} |E[C_i^{2vc}]| + \sum_{1 \leq i \leq t} |C_i^{2vc}|^2 \leq m\sqrt{n} + \sum_{1 \leq i \leq t} |C_i^{2vc}|^2$ and $O(\sum_{1 \leq i \leq t} |C_i^{2vc}|^2) = O(n^2)$ by Lemma 8.3. \square

Lemma 8.3. *Let $G = (V, E)$ be a directed graph and let $C_1^{2vc}, C_2^{2vc}, \dots, C_t^{2vc}$ be the 2-vccs of G . Then $\sum_{1 \leq i \leq t} |C_i^{2vc}| < 3n$.*

Proof. We construct a new graph $G_c = (V_c, E_c)$ from G called a component graph as follows. For each 2-vcc C_i^{2vc} of G , we add a vertex v_i to V_c . Let C_i^{2vc}, C_j^{2vc} be distinct 2-vccs of G . If C_i^{2vc}, C_j^{2vc} have a vertex w in common, then we add a vertex w_* to V_c and two undirected edges $\{v_i, w_*\}, \{w_*, v_j\}$ to E_c . Since G_c is a tree or a forest, $\sum_{1 \leq i \leq t} |C_i^{2vc}| \leq |V| + |E_c| \leq n + n + t - 1 < 3n$. \square

Problem 8.4. *Given a strongly connected graph $G = (V, E)$, find a minimum cardinality set $E^* \subseteq E$ such that the 2-vccs of G coincide with the 2-vccs of the directed graph $G^* = (V, E^*)$ and G^* is strongly connected.*

Lemma 8.5. *There is an $5/3$ approximation algorithm for problem 8.4 with running time $O(nm)$.*

Proof. If we contract the 2-vccs of G that overlap into a super vertex, then we obtain a directed graph, which we call the coarsened graph of G . The edge sets within the 2-vccs of G and the edge set between 2-vccs of G are disjoint. We split the approximation problem 8.4 into two independent problems: problem 8.1 and the minimum strongly-connected spanning subgraph problem. In 2003, Zhao et al. [14]

gave a linear time $5/3$ -approximation algorithm for the minimum strongly-connected spanning subgraph problem. We run this algorithm on the coarsened graph of G . \square

Problem 8.6. *Given a directed graph $G = (V, E)$, find a minimum cardinality set $E^* \subseteq E$ such that the 2-vccs of G coincide with the 2-vccs of $G^* = (V, E^*)$ and the 2-vccs of the coarsened graph of G coincide with the 2-vccs of the coarsened graph of G^* .*

Lemma 8.7. *There is an 1.5 approximation algorithm for problem 8.6 with running time $O(nm)$.*

Proof. The proof is similar to the proof of Lemma 8.2 (we apply the same method on the graph G and on the coarsened graph of G). \square

9. Open problems

We leave as an open problem whether the 2-vccs of a directed graph that contain a certain vertex can be computed in linear time. Another open problem is whether the computing of 3-vccs of a directed graph can be done in $O(nm)$ time.

Acknowledgements.

The Author would like to thank Martin Dietzfelbinger for helpful comments and interesting discussions.

References

- [1] S. Alstrup, D. Harel, P.W. Lauridsen, M. Thorup, Dominators in linear time. SIAM J. Comput. 28(6) (1999) 2117–2132.
- [2] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, J. R. Westbrook, Linear-time algorithms for dominators and other path-evaluation problems, SIAM J. Comput. 38(4) (2008) 1533–1573.
- [3] N. Beldiceanu, P. Flener, X. Lorca, The tree constraint, in: Proc. 2nd Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2005, in: LNCS, vol. 3524, Springer-Verlag, 2005, pp. 64–78.

- [4] J. Cheriyan, R. Thurimella, Approximating minimum-size k -connected spanning subgraphs via matching, *SIAM J. Comput.*, 30(2) (2000) 528–560.
- [5] R. Diestel, *Graph Theory*, 2nd ed., Springer, New York, 2000, pp. 43–44.
- [6] Y. M. Erusalimskii, G. G. Svetlov, Bijoin points, bibridges, and biblocks of directed graphs, *Cybernetics and Systems Analysis*. 16 (1980) 41–44.
- [7] D. Firmani, G. F. Italiano, L. Laura, A. Orlandi, F. Santaroni, Computing strong articulation points and strong bridges in large scale graphs, *SEA, LNCS* 7276, 2012, pp. 195–207.
- [8] L. Georgiadis, Testing 2-vertex connectivity and computing pairs of vertex-disjoint s - t paths in digraphs, in: *Proc. 37th ICALP*, 2010, pp 738–749.
- [9] L. Georgiadis, Approximating the smallest 2-vertex connected spanning subgraph of a directed graph, in: *Proc. 19th European Symposium on Algorithms*, 2011, pp. 13–24.
- [10] H. N. Gabow. Using expander graphs to find vertex connectivity. *J.ACM*, 53 (2006) 800–844.
- [11] M. R. Garey, D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.
- [12] G. F. Italiano, L. Laura, F. Santaroni, Finding strong bridges and strong articulation points in linear time, *Theoretical Computer Science*. 447 (2012) 74–84.
- [13] R. E. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Comput.* 1(2) (1972) 146–160.
- [14] L. Zhao, H. Nagamochi, T. Ibaraki, A linear time $5/3$ -approximation for the minimum strongly-connected spanning subgraph problem, *Information Processing Letters*. 86 (2003) 63–70.